Assignment on Memory Management

K.R. M.M. Kahandawa - UWU/CST/20/041

(1) Paging and segmentation are both memory management techniques used in operating systems to facilitate the conversion of virtual addresses to physical addresses. Each has its own approach to address translation structures, and they differ in terms of the amount of memory required for these structures.

1. Paging

* Address translation structure - paging uses a page table to map virtual addresses to physical addresses. The page table is organized into fixed size ~~bbb~~ blocks or pages, which correspond to fixed size portions of the virtual address space.

* Memory requirements - The size of page table depends on the size of the virtual address space and the page size. If the virtual addresses space is large, a significant portion of memory may be needed to store the page table. However, the page table is relatively simple and doesn't require a large number of entries for most practical applications.

* Fragmentation - Paging can lead to internal fragmentation because pages are typically of a fixed size. This means that some memory within

a page may remain unused, resulting in wasted space.

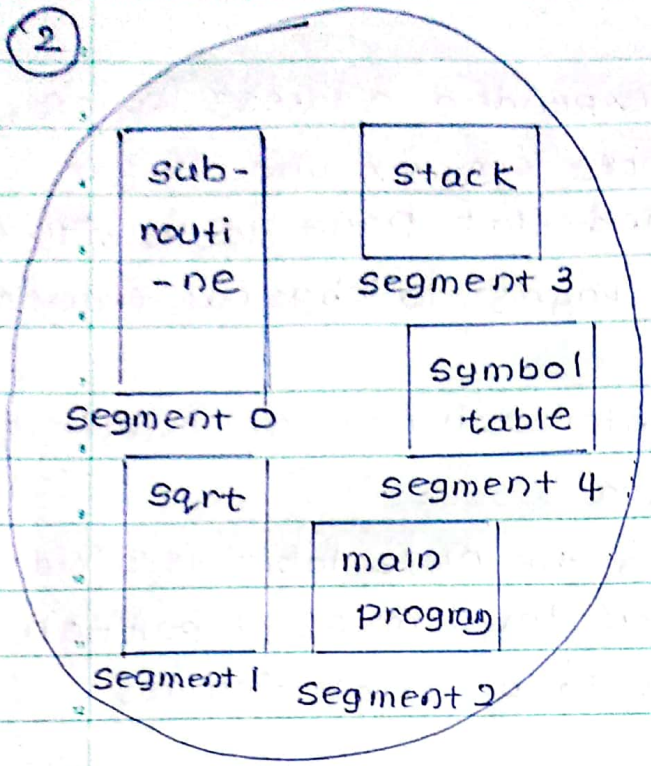
2. Segmentation.

* Address translation structure - Segmentation uses segments to divide the virtual addresses space into different logical segments, such as a code, data and stack. Each segment can have its own base address and size. To translate a virtual address the OS needs to look up the appropriate segment descriptor.

* Memory Requirements - The size of the address translation structures in segmentation depends on the number of segments and their sizes. If there are many segments, or if the segments are of varying sizes, the memory requirements for the segment description can be significant.

* Fragmentation - Segmentation can lead to external fragmentation because segments are not of fixed size. As segments are allocated, gaps may appear in memory that are too small to hold a new segment but too ~~long~~ large to be used efficiently.

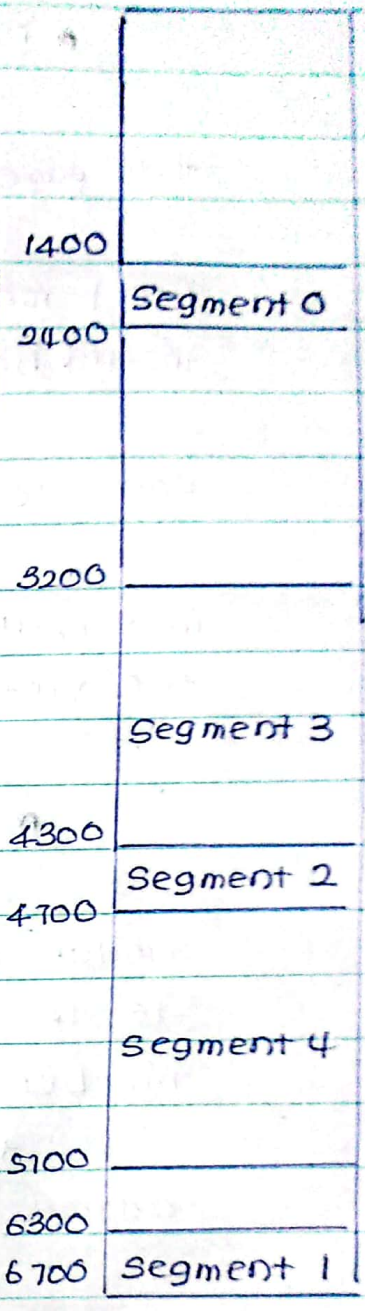
(2)



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment Table

logical address space.



* Principles of Segmentation Memory Management

* principles of Segmentation Management

01. Segmentation Units - Memory is divided into segments. where each segment represents a logical unit such as code segment, data segment or stack segment.

02. Segment Descriptor - Each segment is associated with a segment descriptor, which contains information about the segment's location, size & access rights.

03. Address Translation - When a program reference a memory location, a segment selector is used to locate the corresponding segment descriptor.

04. Segment sharing - Segmentation can facilitate sharing of memory segments among processe. Multiple processes can reference the same segment descriptor, promoting efficient memory utilization.

(3) Sharing a reentrant module is genarally easier when segmentation is used compared to pure paging. This is because segmentation allows for finer-grained control over memory access and sharing, while pure paging tends to be less flexible in this regard.

• Scenario → Consider a multi-user operating system where multiple processes need to execute a common system library or shared code module that is reentrant. This shared code module is used by various process simultaneously.

* Segmentation approach:-

1) Segmentation divides memory logically - In segmentation, different parts of the address space are deivided into segments, each with its own base address and size. you can have a dedicated segment for the shared code module.

2) Shared code segment - Create a shared code segment specifically for the reentrant module. All processes that need to access this module refer-ence the same shared code segment in their segment tables.

3) Memory protection - Can set appropriate access permissions for the shared code segment to ensure that multiple processes can read and execute the code simulteneously, but they can not modify it. this is essential for safety in reentrant modules

4) Efficient sharing - Since all processes refer to the same shared code segment, there is no duplication of code in memory. This make efficient use of memory and any updates to the module are immediately visible to all processes that use it.

* Pure Paging approach :-

1) Paging divides memory into fixed-size pages - memory is divided into fixed-size pages, and each page may contain different parts of code and data.

2) Shared code in page - To share the reentrant module in a paging system, might need to divide the module into pages and allocate these pages to each process. However this can be inefficient and complex, especially if the module span multiple pages.

3) Duplication - Since each progress gets its own copy of the pages containing the reentrant module, this can lead to memory duplication. Multiple instances of the same code module occupy separate memory pages, leading to increased memory usage

4) Synchronization - Ensuring safe concurrent execution of the shared module might require additional synchronization mechanisms to

prevent race conditions and data corruption, as processes may execute their own copies of the module.

* Sharing a reentrant module is easier and more efficient with segmentation, because it allows for the creation of a dedicated shared code segment, efficient memory utilization, and fine-grained control over access permissions.

(4) * Paging in 32-bit Architecture.

- Address space
  * The CPU can generate 32 bit virtual addresse
  * There are 2^{32} (4 GB) possible virtual address

- Page size
  * Typical page size in 32-bit systems are 4 KB (4096 bytes)
  * The virtual address space is divided into pages and the physical memory is divided into page frames of the same size.

- Page Table
  * To map virtual addresses to physical addresses a page table is used.
  * Page table typically consist of multiple entries one for each page in the virtual address space
  * Each entry contains the mapping information for a virtual page to a physical page frame.
  * The page table itself is stored in physical memory.

- Translation process.
  * To translate virtual address to physical address ;
    -> The CPU splits the virtual addresses into a page number and an offset within the page.
    -> It looks up the page number in the page table to find the corresponding page frame number.
    -> The offset within the page is added to the base address of the page frame to form the physical address.

- Memory protection.
  * Paging allows for memory protection by setting access permisions on a per-page basis in the page table entries.
  * Unauthorized accesses result in page faults and are handled by the operating system.

* Paging in 64 bit architecture

- Address space.
  * In 64 bit architecture, the CPU can genarate 64 bit virtual addresses.
  * This provides an extremly large address space of 2^{64}.

- Page size
  * page size in 64 bit systems are typically still 4 KB but can be larger, such as 2 MB or 1 GB.
  * Larger page sizes can reduce the overhead of managing a large number of small pages.

- Page Table

* With the vastly expended address space, the page Table becomes more complex and larger.

* A straightforward flat page table with a 1 to 1 mapping of virtual pages to physical frames would be impractical.

* To manage this, hierarchical or multilevel page tables are often used.

* In these systems, the page table is divided into multiple levels, and each level maps a portion of the virtual address space to physical frames.

- Translation process.

* The translation process in 64-bit system is similar to 32-bit system but involves traversing multip-le levels of page tables to find the physical frame number.

* This hierachical approach helps manage the vast address space more efficiently.

- Memory protection

* remains a key feature in 64-bit systems, allowing for fine-grained control of permissions on a per-page basis, just like in 32-bit systems.

- Large pages.

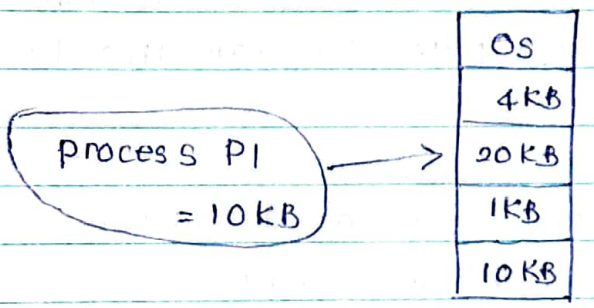
* 64-bit architectures often support large page sizes for improved performance in certain scenarios.

* Large pages can reduce the size of the page table and improve TLB efficiency.

(5) a) Partition Allocation Methods.

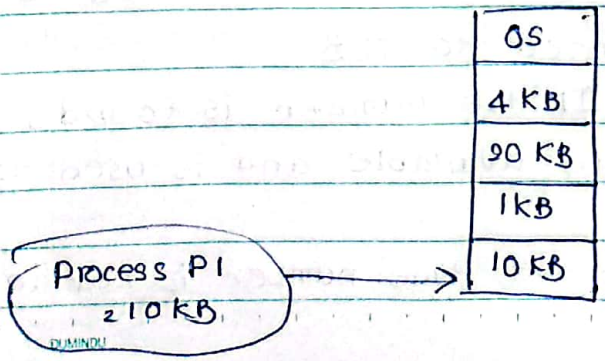
In here, when there is more than one partition freely available to accommodate a processe's request, a partition ~~, a partition allocation~~ must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation. There are different Placement Algorithm;

1. first fit - The partition is allocated which is the first stufficient block from the top of main memory. It scans memory from the begining and change the first available block that is large enough. Thus it allocates the first hole that is large enough.



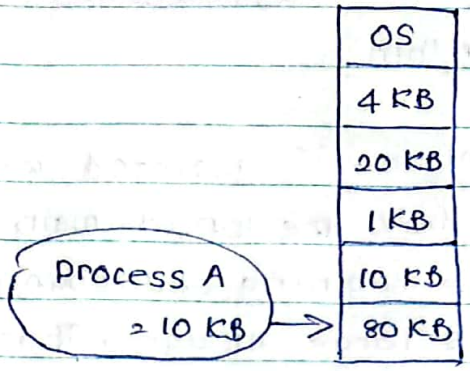
Hole = 20KB - 10KB = 10KB

2. Best fit - Allocate the process to the partition ~~with~~ which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



Hole = 10 KB - 10KB = 0

3. Worst fit - Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is oppsite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.



Hole = 80KB - 10KB = 70 KB.

4. Next fit - This is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

b) Paging and Translation Look-aside Buffer.

* It is fast lookup hardware cache.
* It contains the recently or frequently used page table entries.
* It has 2 parts - Key & Value.
* More expensive.

Paging with TLB

* When a logical address is generated by CPU, its page number is presented to TLB.
* TLB hit - If the number is found, its frame number is immediately available and is used to access memory.
* TLB miss - If the page number is not in the TLB,

a memory reference to the page table must be made

* Hit ratio - precentage of times that a particular page is found in the TLB.